# Project, crustcrawler

fridamei

December 2021

## 1 Controller

The following controller was implemented for controlling the crustcrawler simulation (PD-controller with gravity compensation):

$$u = K_p\tilde{q} - K_d\dot{q} + g \tag{8.51}$$

where $K_p$ and $K_d$ are positiv definite matrices with $k_{p1}, k_{p2}...k_{p6}$ and $k_{d1}, k_{d2}...k_{d6}$ along the diagonals. For some reason, the sign of the gravity matrix had to be swapped to correctly compensate for gravity.
I used dynamic reconfigure to set the desired q-values and $K_p$ and $K_d$ values in order for the manipulator to adjust on-line in the *parameter_callback* function. They are thus defined in the cfg-file. The joint efforts were calculated using the JointState for getting the $\dot{q}$ and $q$ vectors, and using the formula stated above (with swapped sign for the g-matrix). The efforts were then published to the *joint_controller/command* topic.

For the last three joints, joint velocities got very high and I could not achieve adequate control. They oscillated quite a bit before settling without reaching the desired set-point. $K_p = 15$ and $K_d = 7$ seems to give good response for the first three joints, while $K_d$ was set to $< 0.1$ for the remaining three joints. $K_p$ was set to 7 for the fourth joint and 0.7 for the fifth joint. The manipulator broke when trying to control the last joint.

Attached is a video file with a short demonstration of the controller, and following is the controller code. The workspace as a whole is attached in a zipped folder, containing the cfg file and launch files.

To run the code, the crustcrawler simulator and crustcrawler lib repositories must be added to the workspace.

PD-controller code (C++):

```
#include <ros/ros.h>
#include <eigen3/Eigen/Eigen>
#include <sensor_msgs/JointState.h>
```

```
#include <crustcrawler_lib/dynamics_simple_6dof.h>  // For the gravity model
#include <std_msgs/Float64.h>

// For the dynamic reconfigure (changing the K_d, K_p and q_d on-line)
#include <dynamic_reconfigure/server.h>
#include <joint_space_controller/PdControlConfig.h>

// Joints (global to avoid sending as argument to all functions using it)
Eigen::VectorXd q(6);
Eigen::VectorXd q_d(6);
Eigen::VectorXd q_dot(6);
Eigen::VectorXd K_p(6);
Eigen::VectorXd K_d(6);

// 6x6 Identity matrix if making the k_p and k_d as having equal diagonals
// Eigen::Matrix6d I = Eigen::Matrix6d::Identity();

// Get the gravity vector for the manipulator model. Located in crustcrawler_lib/dynamics_simple_6dof.cpp (getGravityVector(q))
crustcrawler_lib::DynamicsSimple6DOF simple_dynamics;

// From the controller written in tutorial and mandatory assignment 1; publish to a global joint command then update the local joint_command
ros::Publisher* command1_pub_global = NULL;
ros::Publisher* command2_pub_global = NULL;
ros::Publisher* command3_pub_global = NULL;
ros::Publisher* command4_pub_global = NULL;
ros::Publisher* command5_pub_global = NULL;
ros::Publisher* command6_pub_global = NULL;

// For future ref.: https://www.youtube.com/watch?v=YKZkZSVcsnI
// Dynamic reconfigure (update the parameters on-line)
void parameter_callback(joint_space_controller::PdControlConfig& config, uint32_t level) {
  ROS_INFO("New values: \nDesired joint values: [%.2f], [%.2f], [%.2f], [%.2f], [%.2f], [%.2f]\n"
    "K_p: [%.2f], [%.2f], [%.2f], [%.2f], [%.2f], [%.2f]\n"
    "K_d: [%.2f], [%.2f], [%.2f], [%.2f], [%.2f], [%.2f]\n",
    config.groups.q_d.q1_d, config.groups.q_d.q2_d, config.groups.q_d.q3_d, config.groups.q_d.q4_d, config.groups.q_d.q5_d, config.groups.q_d.q6_d,
    config.groups.k_p.k1_p, config.groups.k_p.k2_p, config.groups.k_p.k3_p, config.groups.k_p.k4_p, config.groups.k_p.k5_p, config.groups.k_p.k6_p,
    config.groups.k_d.k1_d, config.groups.k_d.k2_d, config.groups.k_d.k3_d, config.groups.k_d.k4_d, config.groups.k_d.k5_d, config.groups.k_d.k6_d);

  K_p(0) = config.groups.k_p.k1_p;
  K_p(1) = config.groups.k_p.k2_p;
  K_p(2) = config.groups.k_p.k3_p;
  K_p(3) = config.groups.k_p.k4_p;
```

```
  K_p(4) = config.groups.k_p.k5_p;
  K_p(5) = config.groups.k_p.k6_p;

  K_d(0) = config.groups.k_d.k1_d;
  K_d(1) = config.groups.k_d.k2_d;
  K_d(2) = config.groups.k_d.k3_d;
  K_d(3) = config.groups.k_d.k4_d;
  K_d(4) = config.groups.k_d.k5_d;
  K_d(5) = config.groups.k_d.k6_d;

  q_d(0) = config.groups.q_d.q1_d;
  q_d(1) = config.groups.q_d.q2_d;
  q_d(2) = config.groups.q_d.q3_d;
  q_d(3) = config.groups.q_d.q4_d;
  q_d(4) = config.groups.q_d.q5_d;
  q_d(5) = config.groups.q_d.q6_d;
}

// Get the current manipulator joint values
void joint_state_callback(const sensor_msgs::JointState::ConstPtr&
    msg) {
  // Get the current joint positions and velocities
  for (int i = 0; i < 6; i++) {
    q(i) = msg->position[i];
    q_dot(i) = msg->velocity[i];
  }
  // Error vector
  Eigen::VectorXd q_e(6);

  // Get the gravity component from the crustcrawler_lib package
  Eigen::VectorXd g(6);
  g = simple_dynamics.getGravityVector(q);

  // Calculate the position error
  q_e = q_d - q;

  ROS_INFO("Error: [%.2f], [%.2f], [%.2f], [%.2f], [%.2f], [%.2f]\n
    ", q_e(0), q_e(1), q_e(2), q_e(3), q_e(4), q_e(5));

  // Control effort: u = Kp*q_tilde - Kdq_dot + g (Equation 8.51 in
     Siciliano)
  Eigen::VectorXd u(6);
  u = K_p.cwiseProduct(q_e) - K_d.cwiseProduct(q_dot) + g;

  // Must publish correct type (Float64) in order to avoid typing
    errors in main where local variable is set to this global
    variable.
  // The joint_controller/command topics are of type Float64
  std_msgs::Float64 output;

  // Joint 1
  output.data = u(0);
  command1_pub_global->publish(output);

  //Joint 2
  output.data = u(1);
  command2_pub_global->publish(output);
```

```
98
99    // Joint 3
100   output.data = u(2);
101   command3_pub_global->publish(output);
102
103   // Joint 4
104   output.data = u(3);
105   command4_pub_global->publish(output);
106
107   // Joint 5
108   output.data = u(4);
109   command5_pub_global->publish(output);
110
111   // Joint 6
112   output.data = u(5);
113   command6_pub_global->publish(output);
114
115   /*
116   Check that the node connects properly to the manipulator
117   ROS_INFO_THROTTLE(1, "Receiving message");
118   Check that the joint states are published by the crustcrawler:
        rostopic echo crustcrawler/joint_states
119   */
120 }
121
122 int main(int argc, char** argv) {
123   ros::init(argc, argv, "pd_controller");
124   ros::NodeHandle nh;
125
126   //Initial values
127   float n = 0.0;
128   float m = 0.0;
129   K_p.setConstant(n);
130   K_d.setConstant(m);
131   q_d << 0.0, 0.0, 0.0, 0.0, 0.0, 0.0;
132
133   // Subscribe to the joint states from the crustcrawler
134   ros::Subscriber joint_state_sub = nh.subscribe("crustcrawler/
        joint_states", 1000, &joint_state_callback);
135
136   // The joint commands are different topics, need one publisher
        for each one
137   // Publish the joint commands
138   ros::Publisher command1_pub = nh.advertise<std_msgs::Float64>("
        crustcrawler/joint1_controller/command", 1000);
139   ros::Publisher command2_pub = nh.advertise<std_msgs::Float64>("
        crustcrawler/joint2_controller/command", 1000);
140   ros::Publisher command3_pub = nh.advertise<std_msgs::Float64>("
        crustcrawler/joint3_controller/command", 1000);
141   ros::Publisher command4_pub = nh.advertise<std_msgs::Float64>("
        crustcrawler/joint4_controller/command", 1000);
142   ros::Publisher command5_pub = nh.advertise<std_msgs::Float64>("
        crustcrawler/joint5_controller/command", 1000);
143   ros::Publisher command6_pub = nh.advertise<std_msgs::Float64>("
        crustcrawler/joint6_controller/command", 1000);
144
145   command1_pub_global = &command1_pub;
```

```
146   command2_pub_global = &command2_pub;
147   command3_pub_global = &command3_pub;
148   command4_pub_global = &command4_pub;
149   command5_pub_global = &command5_pub;
150   command6_pub_global = &command6_pub;
151
152   dynamic_reconfigure::Server<joint_space_controller::
        PdControlConfig> server;
153   dynamic_reconfigure::Server<joint_space_controller::
        PdControlConfig>::CallbackType f;
154   f = boost::bind(&parameter_callback, _1, _2);
155   server.setCallback(f);
156
157   //Debugging: for printing matrices:
158   //std::cout << matrix << std::endl;
159
160   // Execute subsriber callbacks
161   ros::spin();
162
163   return 0;
164 }
```

Listing 1: Controller node